Amendments to the Claims:

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

Claim 1 (currently amended) A lock contention management method, comprising:

determining whether a code module ~~should continually request~~ has failed to acquire access to a lock to keep other code modules from accessing a resource; ~~or should stop requesting access to the lock; and~~

changing from an original priority of the code module to a lower priority if the code module should continually request access, and allowing the code module to continue to request access at ~~do so at a lowered~~ the lower priority if it is determined that there are no other code modules waiting to run; and

restoring the code module to its original priority after the code module either obtains the lock or is forced to sleep after some fixed period of time.

Claim 2 (original) The method of claim 1, wherein:

the code module continually requests access to the lock when the resource has other tasks to run; and

the code module stops requesting access to the lock when the resource has no other tasks to run.

Claim 3 (original) The method of claim 1, further comprising determining that there are multiple processor run queues and having the code module stop requesting access to the lock if there are other code modules in the multiple processor run queues waiting to access the lock and having the code module continually request access to the lock if there are not.

Claim 4 (original) The method of claim 1, further comprising determining that there is a single processor run queue and having the code module stop requesting access to the lock if there are other code modules in the single processor run queue

waiting to access the lock and having the code module continually request access to the lock if there are not.

Claim 5 (original) The method of claim1, wherein the code module stops requesting access to the lock using a low-priority execution technique that lowers a priority of the code module.

Claim 6 (original) The method of claim 5, wherein the low-priority execution technique allows a higher-priority code module to access the lock first.

Claim 7 (original) The method of claim 6, wherein the lowered-priority code module continually requests access to the lock if no higher-priority code modules are available.

Claim 8 (canceled).

Claim 9 (original) The method of claim 8, further comprising restoring a priority of the code module to the original priority.

Claim 10 (original) The method of claim 9, wherein the original priority is restored after a specified period of time.

Claim 11 (currently amended) A lock contention management system, comprising:

a dispatch management module that ~~determines~~ undispatches a code module when ~~a~~ the code module ~~should wait for the~~ has failed to acquire a lock ~~by becoming undispatched~~ and other code modules are waiting to run ~~when the code module should try to access the lock by spinning~~; and

a low-priority execution module that lowers a priority of the code module when the code module becomes undispatched;

wherein, if the code module continually requests access and if there are no other code modules waiting to run, changing from an original priority of the code module to the lower priority with the low-priority execution module, dispatching the code module with the dispatch management module and allowing the code module to spin at the lower priority;

wherein the code module is restored to its original priority after the code module either obtains the lock or is forced to sleep after some fixed period of time.

Claim 12 (original) The lock contention management system of claim 11, wherein the dispatch management module determines that the code module should spin when a processor has no other tasks to perform.

Claim 13 (original) The lock contention management system of claim 11, wherein the dispatch management module determines that the code module should become undispatched when a processor has other tasks to perform.

Claim 14 (original) The lock contention management system of claim 11, wherein the code module is a program thread.

Claim 15 (original) The lock contention management system of claim 11, wherein the low-priority execution module reduces an original priority of the code module to a lowered priority.

Claim 16 (canceled).

Claim 17 (original) The lock contention management system of claim 15, wherein the low-priority execution module allows higher-priority code modules to acquire the lock before the lower priority code module.

Claim 18 (currently amended) A method of acquiring a lock to allow execution of a program thread by a computer processor, comprising:

undispatching the program thread when the program thread has failed to acquire a lock and other program threads are waiting to run;

~~determining whether to have the program thread to spin or become undispatched while waiting to acquire the lock; and~~

lowering a priority of the program thread before spinning or undispatching;

dispatching the program thread and changing from an original priority of the program thread to the lower priority if the program thread continually requests access and if there are no other program threads waiting to run and allowing the code module to spin at the lower priority; and

restoring the program thread to its original priority after the program thread either obtains the lock or is forced to sleep after some fixed period of time.

Claim 19 (currently amended) The method of claim 18, wherein the program thread is allowed to spin if there are no other program threads waiting to access the lock and if the computer processor does not have other processing to perform.

Claim 20 (currently amended) The method of claim 18, wherein the program thread ~~is allowed to~~ becomes undispatched if there are other program threads waiting to access the lock and if the computer processor has other processing to perform.